Copyright © 2018
Luca Grella & Daniele Lunghi
All rights reserved

DESIGN DOCUMENT

TRACKME – VERSION 1.0 – 10/12/2018

Luca Grella 905655 & Daniele Lunghi 905083
SOFTWARE ENGINEERING 2 PROJECT

1. INTRODUCTION

1A. Purpose

This document integrates the RASD with a more detailed description of the system, its focus is on the technical side of the problem, in order to provide the developers with a detailed description of the critical aspects of the project.
In particular, the following elements are considered:

1. High level architecture, where the interactions between the various components of the system are described.
2. Design models
3. Runtime behaviour, where the interactions of the components, when performing specific tasks, are described.
4. Most relevant design patterns.

1B. Scope

The system consists of two services: Data4Help and AutomatedSOS.
Data4Help is the main one, which allows third part users (TPUs) to access data from normal users (NUs), as described in the RASD.
Because of this, NUs need to have an app on the mobile phone, which will interact through the appropriate APIs with the devices, while TPUs need to be able to interact with a computer, in order to handle the queries and to be provided with an appropriate interface when accessing and handling the data.
We chose to allow both categories to access the system with a mobile application and using a program on a personal computer, which will have different features based on the type of user.
As the main concern of the application is data handling, it needs to operate in respect of the local privacy regulations (like the GDPR in the European Community).
The Application and the computer program allow the administrator to interact directly with the system, managing the data and solving one-time problems, when they occur.
In particular he will have a privileged access to all the data, the power to ban and accept into the system any user and he will be provided with a complete access to any part of the system.
The system includes also the AutomatedSOS functionalities, built on the same servers, which functionalities have been already described in the RASD.

1C. Acronyms

RASD	Requirement Analysis Specification Document
DD	Design Document
UI	User Interface
COTS	Commercial off the Shelf
DBMS	Database Management System

1D. Revision History

During the writing of the DD we modified some parts of the RASD, in order to make the whole project coherent.
The changes made are explained in detail in the second version of the RASD.

1E. Reference Documents

- "Mandatory Project Assignment AY 2018-2019.pdf".

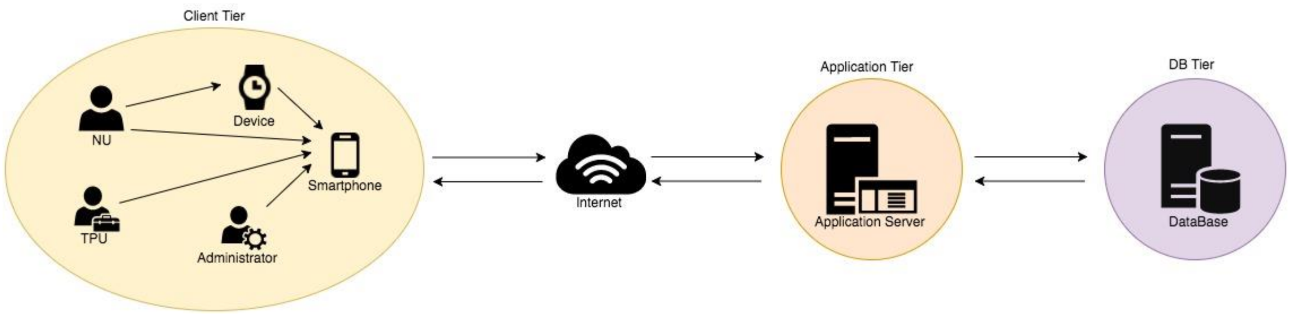
1F. Document Structure

This document is divided into 6 macro-chapters. The first part contains the introduction, the objectives of the project and the indications to better read the document. The second part contains the architectural choices and the various connections between the components explained with diagrams. The third part provides an overview on how the user interface will look like. The fourth part explains how the requirements written in RASD map to the design defined in DD. The fifth part explains the order in which it is planned to implement the subcomponents of the system (and the integration between subcomponents and test integration). The sixth part includes the division of working hours among group members. Finally, the last part cites the references used for the realization of the project.

2. ARCHITECTURAL DESIGN

2A. Overview: High level components and interactions

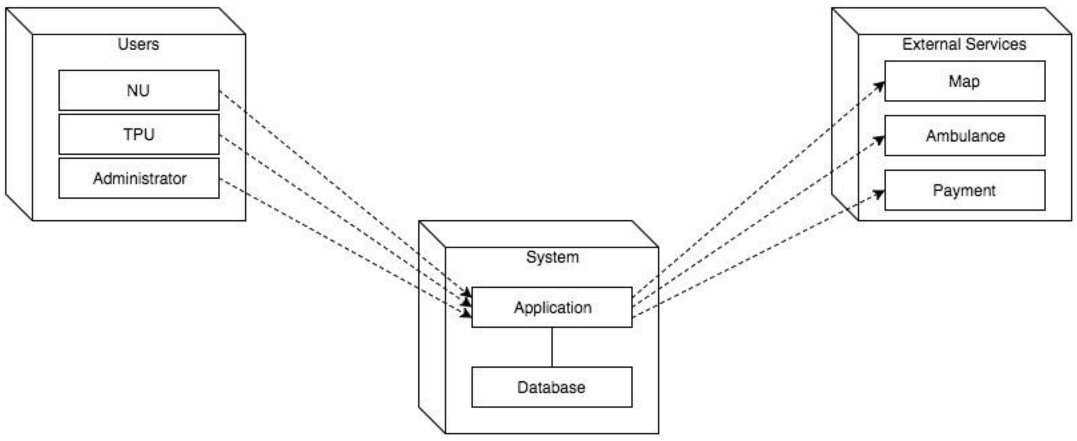
We chose to build Data4Help on a three-level architecture and to implement AutomatedSOS on it.



Client Tier: First level of the system, it handles the devices of the NUs, provides all the users with an interface and interacts with the Application Tier.
Application Tier: Contains the logic of the system, AutomatedSOS is implemented here. It works as a bridge between the client and the database tiers, moving and processing data between the two levels.
It also contains a small local database.
Data Tier: Information are stored here and transferred to the Application Tier for processing or in order to be routed to the users.

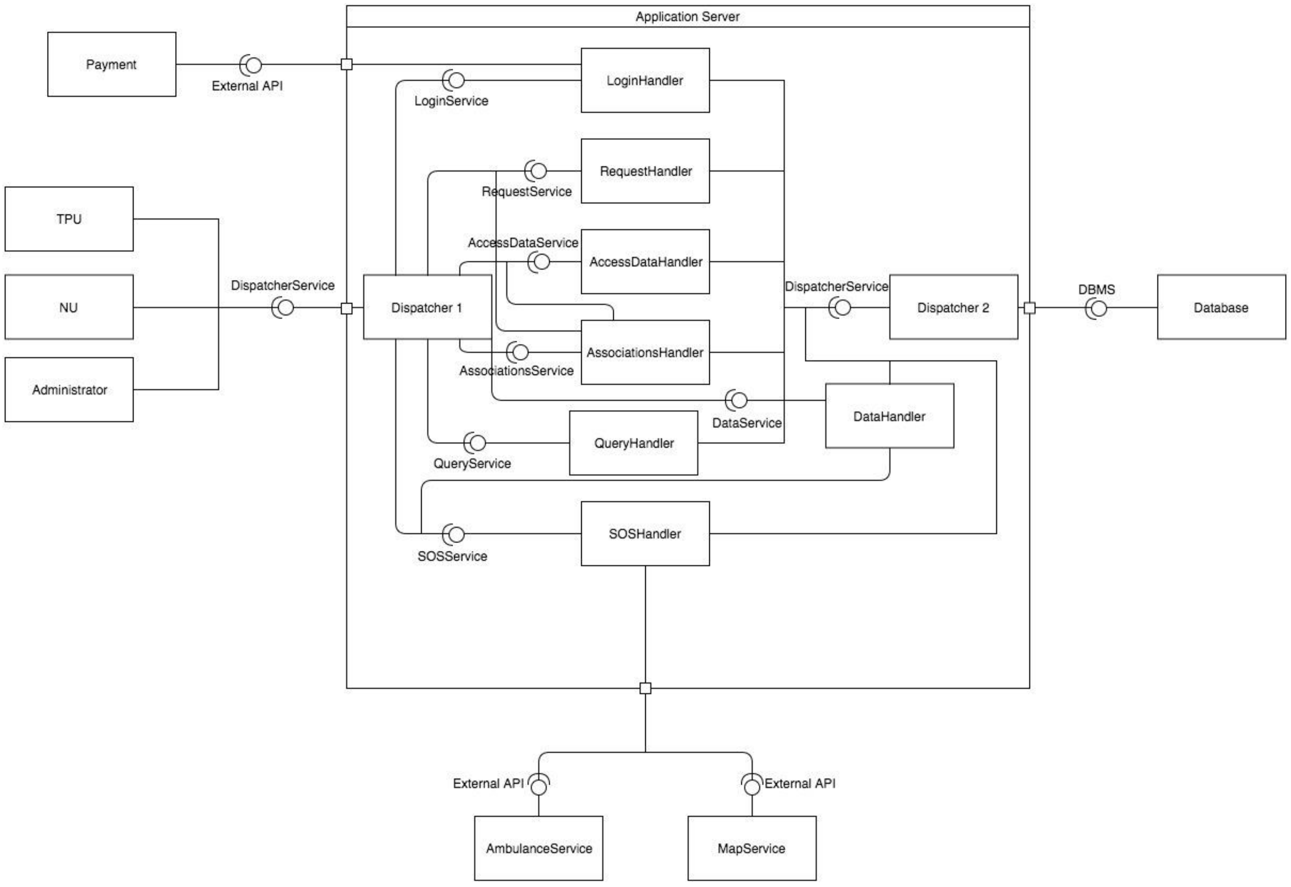
High level components

FAKE
The FAKE consists of two physical structures: Application Server and the Database Server.
The Database manages the data of the entire system and of the customers who use it.
The information contained in it are available to the other components of the system thorough the appropriate interfaces.
The Application Server receives some requests from the customers and it handles, using the data from the Database.
Here is implemented the AutomatedSOS logic, which needs to have a fast access to the most recent data from the users.
Because of that the Application Server is provided with a small, local Database.
MOBILE APPLICATION
The mobile application allows all the users and the administrator to interact directly with the system. It provides them with and interface and it handles the devices of the NUs, guaranteeing that the flow of data they produce is routed to the FAKE.
EXTERNAL SERVICES



2B. Component view

This section is centred on the component diagram and its descriptions in order to show how the components of the architecture are connected to each other to form larger components.
It also shows the system's structure from a software point of view.

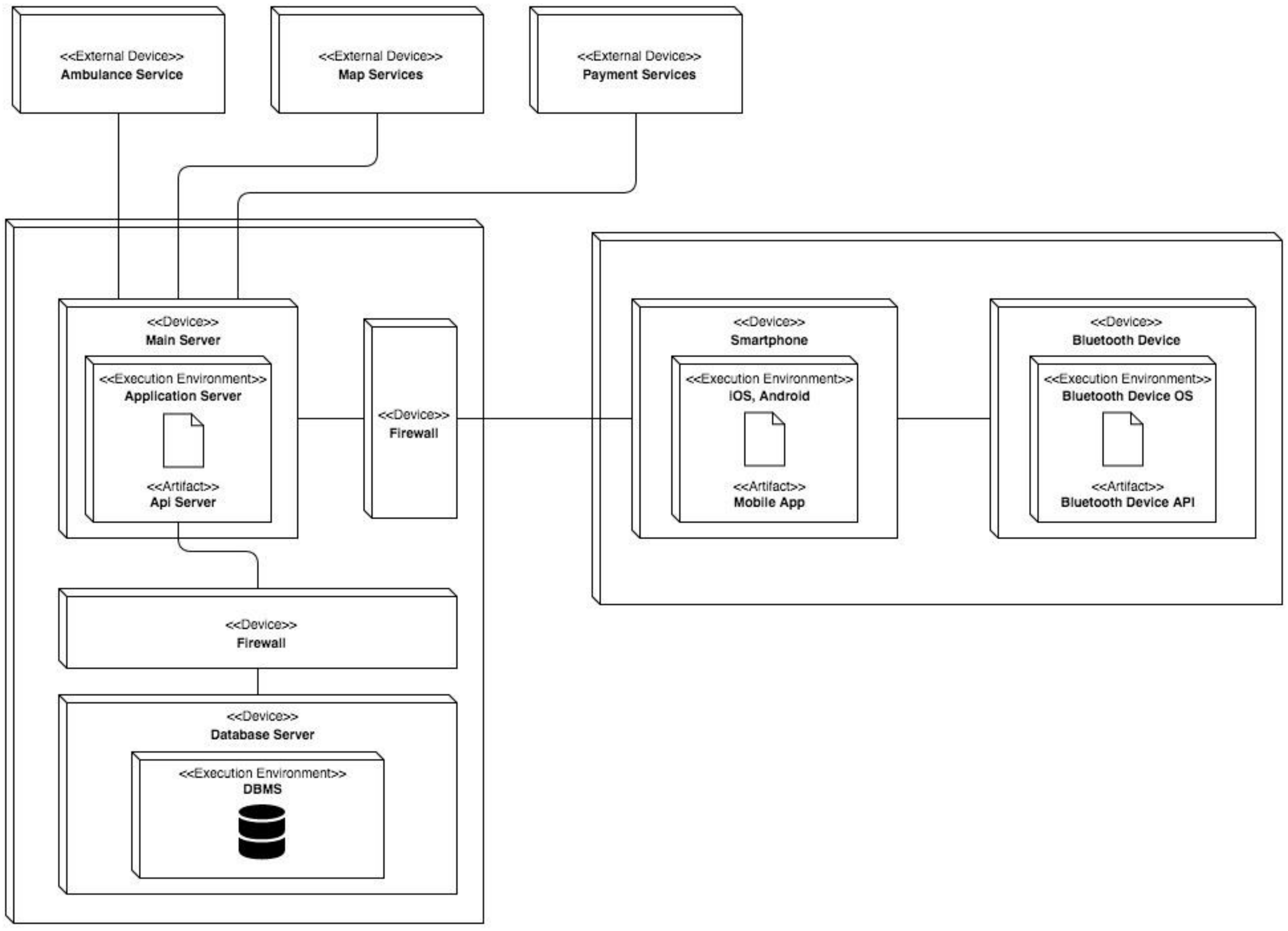


NU: The normal user's mobile app or computer program.
Administrator: The administrator of the service (mobile app and computer program).
TPU: The third part user's mobile app or computer program.
Device: The device connected to the normal user (for example the apple watch).
Dispatcher: It collects and redirects all the requests passing through it.
DataHandler: It receives the data from the first dispatcher, it organizes them and send them to the SOSHandler and to the second dispatcher, which will route them to the Database.
RequestsHandler: It handles the requests from the TPUs to the NUs. It interacts with the Database (where the requests are stored).
AccessDataHandler: It handles the request from the TPUs for the data of individual NUs, checking that there is a previously accepted request sent by the third part to the normal user.
In order to do it, it needs to interact with the AssociationsHandler.
LogInHandler: Component responsible for the Login and Sign Up operations of the users.
It's associated to the PaymentService in order to guarantee the correct management of the AutomatedSOS accounts.
AssociationsHandler: It keeps tracks of the connections between the normal users and the third part users.
It interacts with the Database (where the associations are stored) and it observes the RequestsHandler, in order to know when a request is accepted and create an association.
QueryHandler: It handles the anonymous queries from the Third Part Users
SOSHandler: Here is implemented the logic of AutomatedSOS. it interacts with the MapService and the AmbulanceService, both external to the system.
It also checks the internal state of the DataHandler in order to know if there is an emergency.

DataHandler: It performs the basic operations on the data received from the users before sending them to the Database.
it's provided with a small internal Database.
Database: Here are stored the persistent data of the system
AmbulanceService: The external service responsible for the ambulances in the location of the user.
MapService: The external service responsible of associating the coordinates with a location.

2C. Deployment view

This section shows how the software components are distributed over the hardware resources of the system.
It also provides a basic description of the security measures used, with a particular focus on the use of the firewalls.

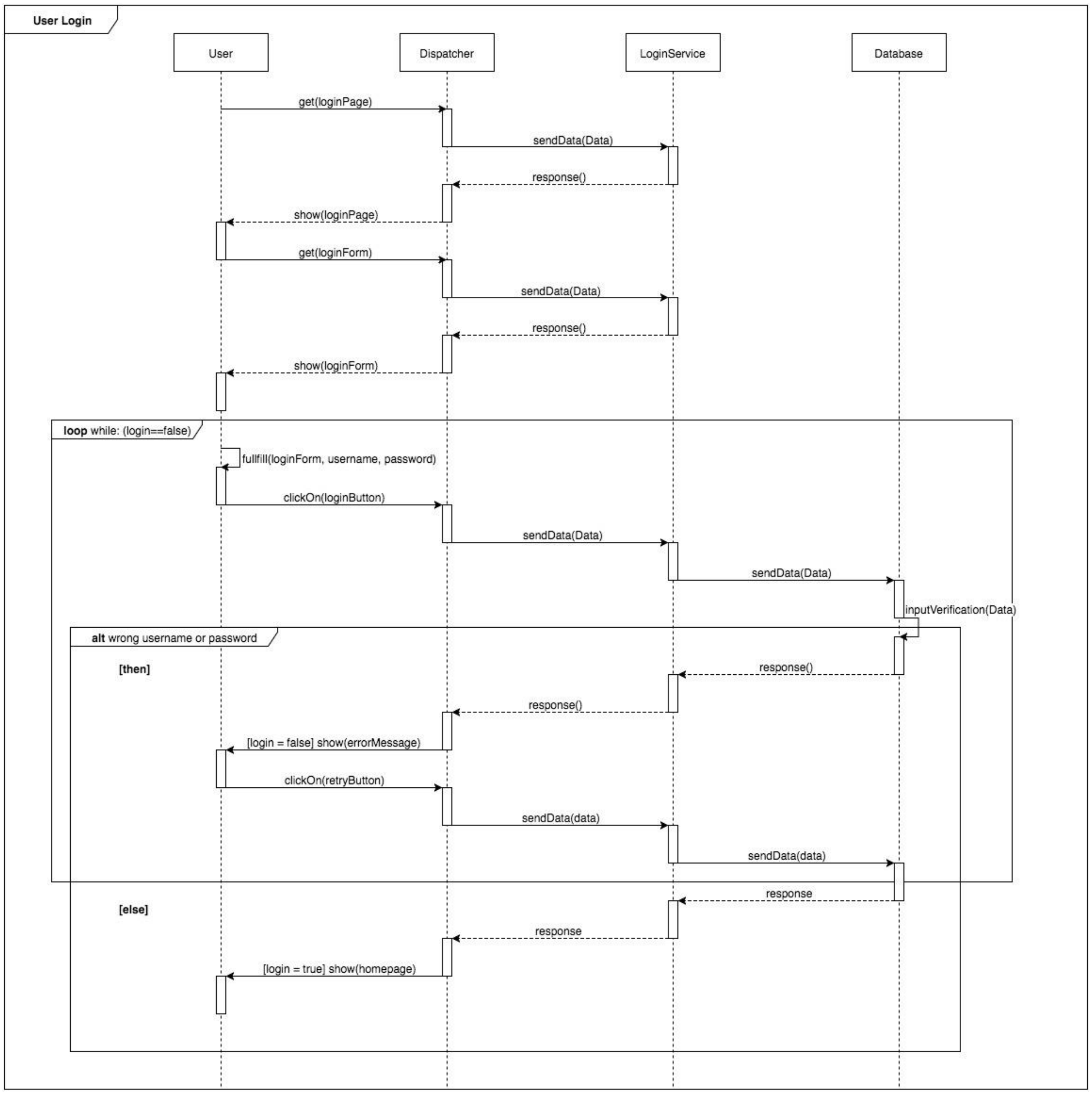


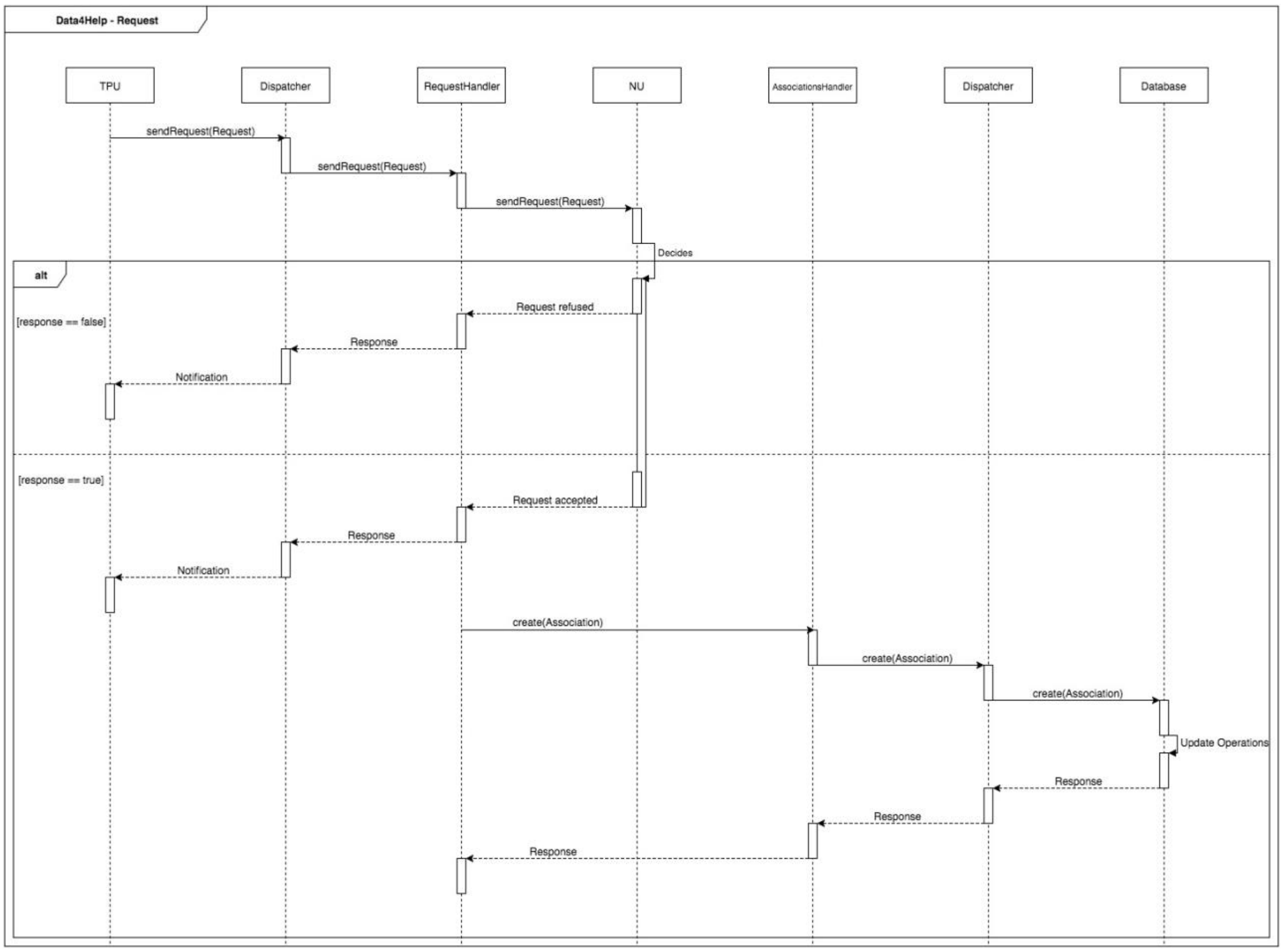
<<External Device>> Ambulance Service
<<External Device>> Map Services
<<External Device>> Payment Services
<<Device>> Main Server
<<Execution Environment>> Application Server
<<Artifact>> Api Server
<<Device>> Firewall
<<Device>> Firewall
<<Device>> Database Server
<<Execution Environment>> DBMS
<<Device>> Smartphone
<<Execution Environment>> IOS, Android
<<Artifact>> Mobile App
<<Device>> Bluetooth Device
<<Execution Environment>> Bluetooth Device OS
<<Artifact>> Bluetooth Device API

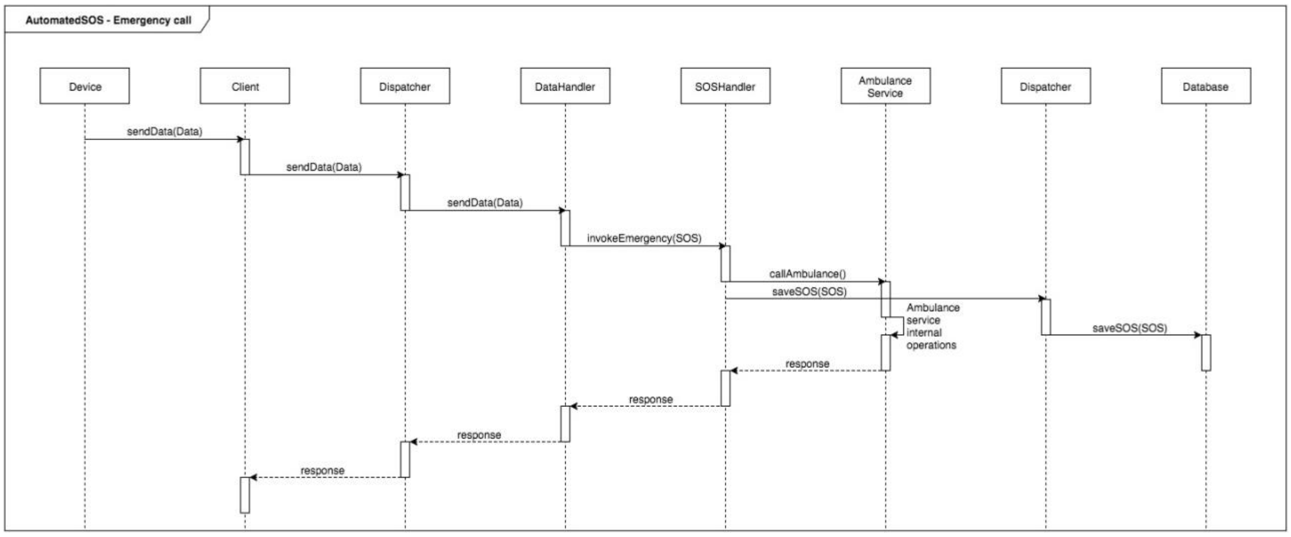
Considering that the application treats sensible data, we decided to place a firewall between the client and the Application Server, and another one between the Application Server and the database.
This guarantees that most of the data will remain secure even if the Application Server has been hacked.

2D. Runtime view

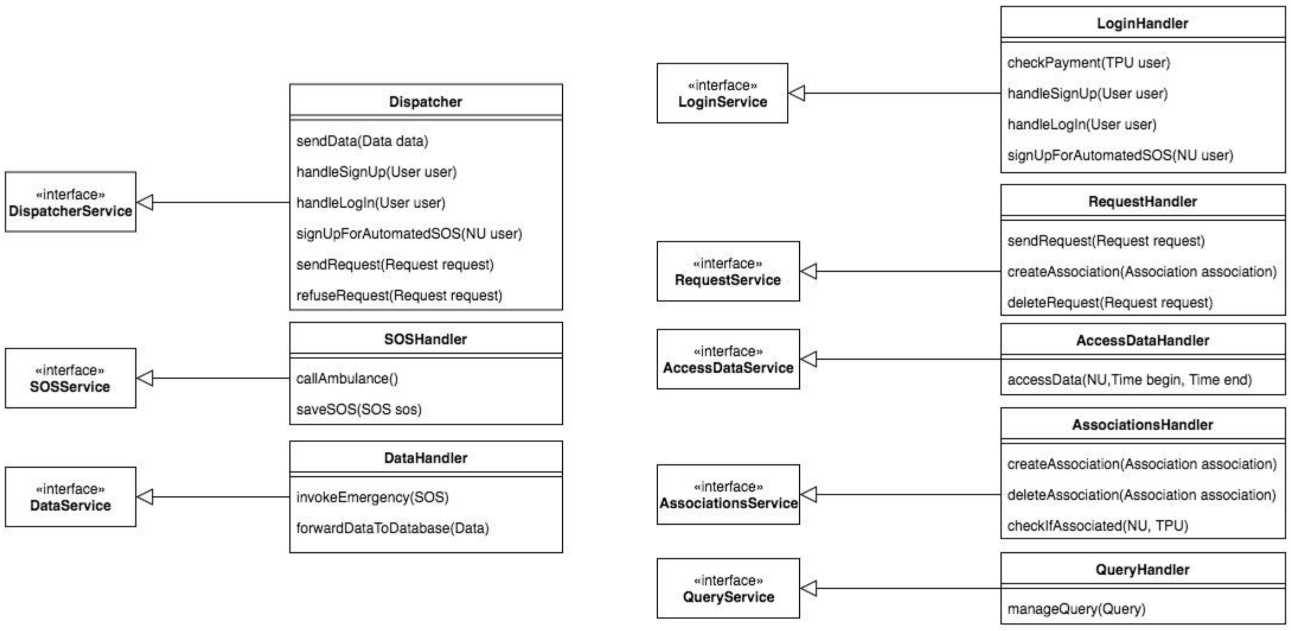
Here sequence diagrams are used in order to describe the interactions of the components in order to perform specific tasks.




Data4Help - Request
TPU
Dispatcher
RequestHandler
NU
AssociationsHandler
Dispatcher
Database
sendRequest(Request)
sendRequest(Request)
sendRequest(Request)
Decides
alt
[response == false]
Request refused
Response
Notification
[response == true]
Request accepted
Response
Notification
create(Association)
create(Association)
create(Association)
Update Operations
Response
Response
Response


AutomatedSOS - Emergency call
Device
Client
Dispatcher
DataHandler
SOSHandler
Ambulance Service
Dispatcher
Database
sendData(Data)
sendData(Data)
sendData(Data)
invokeEmergency(SOS)
callAmbulance()
saveSOS(SOS)
Ambulance service internal operations
saveSOS(SOS)
response
response
response
response

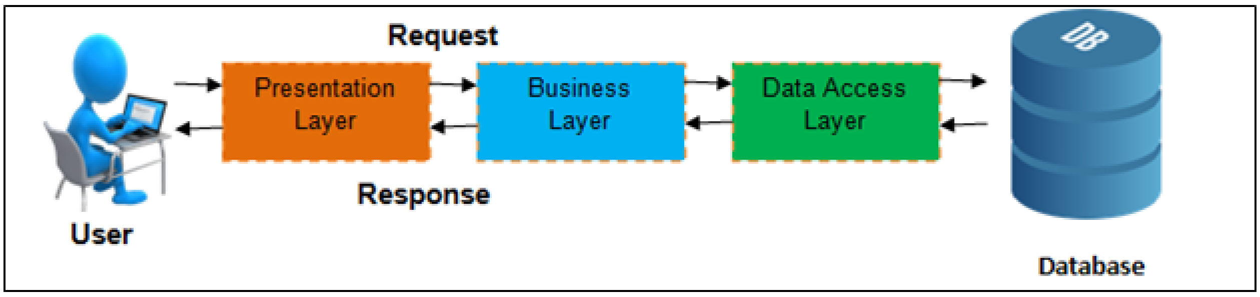
2E. Component interfaces


«interface» DispatcherService
Dispatcher
sendData(Data data)
handleSignUp(User user)
handleLogIn(User user)
signUpForAutomatedSOS(NU user)
sendRequest(Request request)
refuseRequest(Request request)
«interface» SOSService
SOSHandler
callAmbulance()
saveSOS(SOS sos)
«interface» DataService
DataHandler
invokeEmergency(SOS)
forwardDataToDatabase(Data)
«interface» LoginService
LoginHandler
checkPayment(TPU user)
handleSignUp(User user)
handleLogIn(User user)
signUpForAutomatedSOS(NU user)
«interface» RequestService
RequestHandler
sendRequest(Request request)
createAssociation(Association association)
deleteRequest(Request request)
«interface» AccessDataService
AccessDataHandler
accessData(NU,Time begin, Time end)
«interface» AssociationsService
AssociationsHandler
createAssociation(Association association)
deleteAssociation(Association association)
checkIfAssociated(NU, TPU)
«interface» QueryService
QueryHandler
manageQuery(Query)

2F. Selected architectural styles and models

Here are described an explained the architectural choices made during the implementation of the system.

Overall Architecture



Design Decisions

We opted for a COTS solution for the Data Base, we'll buy a database big enough for our data and a DBMS particularly reliable, secure and highly performant, in order to manage the big amount of data in our system in accord to the requirements specified in the RASD.
We will use the API of the DBMS as an interface with which our system can communicate.
The Smart Devices will be handled through the specific APIs, which will allow them to communicate with the client of our application, running on the mobile phone of the user.
In order to guarantee both the need for the Normal Users to have an application running on a mobile and able to connect with the smart devices and the possibility for the Third Part Users to comfortably access the data from the screen of a computer, we will create both a mobile phone application and a computer program, both connected to the Application Service.
However, the NUs will be obliged to use the application in order to connect the devices, while they can choose whether they want to use also the computer program.
The TPUs are free to choose any of the two options, or both.

Therefore, the Dispatcher must provide an interface for both for the application and the computer program requests.

Design Patterns

- <u>MVC</u>: MVC Pattern stands for Model-View-Controller Pattern. This pattern is used to separate application's concerns.
    - Model - The Model component corresponds to all the data-related logic that the user works with. This can represent either the data that is being transferred between the View and Controller components or any other business logic-related data.
    - View - The View component is used for all the UI logic of the application. For example, the Customer view will include all the UI components such as text boxes, dropdowns, etc. that the final user interacts with.
    - Controller - Controllers act as an interface between Model and View components to process all the business logic and incoming requests, manipulate data using the Model component and interact with the Views to render the final output.

In particular, our application will use a PHP server based on MVC logic. Our Mobile interface will use Titanium, an MVC framework, based on alloy language, to enable the rapid development of mobile applications.

-<u>Client-Server</u>: The client-server model is a distributed communication framework of network processes among service requestors, clients and service providers. The client-server connection is established through a network or the Internet. We chose this structure mainly because of security reasons, because it guarantees more protection to the data.

-<u>Observer</u>: Software design pattern in which an object, called the subject, maintains a list of its dependents, called observers, and notifies them automatically of any state changes, usually by calling one of their methods.
We decided to use the Observer Pattern in order to model the interactions between the AssociationsHandler and the RequestsHandler, as the AssociationsHandler observes the requests and creates an association as soon as the request is accepted.
Furthermore the SOSHandler observes the DataHandler in order to be informed when there is an emergency and to notify it to the AmbulanceService.

-<u>Façade</u>: Facade pattern hides the complexities of the system and provides an interface to the client using which the client can access the system. This type of design pattern comes under structural pattern as this pattern adds an interface to existing system to hide its complexities.

In our system this role is covered by the dispatchers, which offers to every component of the system a simplified interface, providing a standard interface for the communications among them.

3. USER INTERFACE DESIGN

The design of the interfaces has been already dealt in the RASD (Section 3A1.)

4. REQUIREMENTS TRACEABILITY

This section explains how the requirements specified in the RASD are related to design elements.

G1. NU's account correct handling.

- LoginHandler

G2. TPU's account correct handling.

- LoginHandler

G3. Users data registration.

- Client
- Dispatchers
- DataHandler
- Database

G4. Allow TPU to access the data of NU, upon acceptance.

- Dispatchers
- RequestsHandler
- AssociationsHandler
- Database

G5. Allow TPU to access anonymous data of a group of at least one thousand people.

- Dispatchers
- QueryHandler
- Database

G6. Notify ill if health values are below threshold for more than 5 seconds.

- Dispatcher 1
- SOSHandler

5. IMPLEMENTATION, INTEGRATION AND TEST PLAN

This last section provides the developers with a planned sequence in which the subcomponents should be implemented.
A testing plan is also made available.
It also shows the order in which we plan to implement the sub-components of our system and the order in which we plan to integrate these sub-components and test the integration.

Considering the complexity of the system and the high number of interactions among most of the components, the integration among them should start as soon as possible.
For the same reason, tests on the interactions among them must be ran in parallel with the tests on the singular components.
The dispatchers may represent the bottleneck of the system, because all the requests from the client and to the Database pass thorough them: because of this they will require a particular attention during the implementation and testing phases, and it's highly probable that they will be the most expensive components of the system.
As mentioned before we opted for a COTS solution for the Data Base, so we only need to implement the connection between it and the Application Server.
The component responsible for this connection will be the Dispatcher, in accord with the general scheme of the system.

Implementation order

In order to build the implementation plan, the following dependencies must be taken into account:

1) The dispatchers are related to every component of the system. Because of this, they should be implemented first, in order to guarantee an interface between each component and the system.
2) The Database, the QueryHandler, the AccessDataHandler and the SOSHandler depend on the data treated by the DataHandler, which should be implemented before them.
3) The RequestHandler and the AssociationsHandler are strongly related, so they should be implemented in parallel. The AccessDataHandler depends on both.

Therefore, the implementation order will be the following:

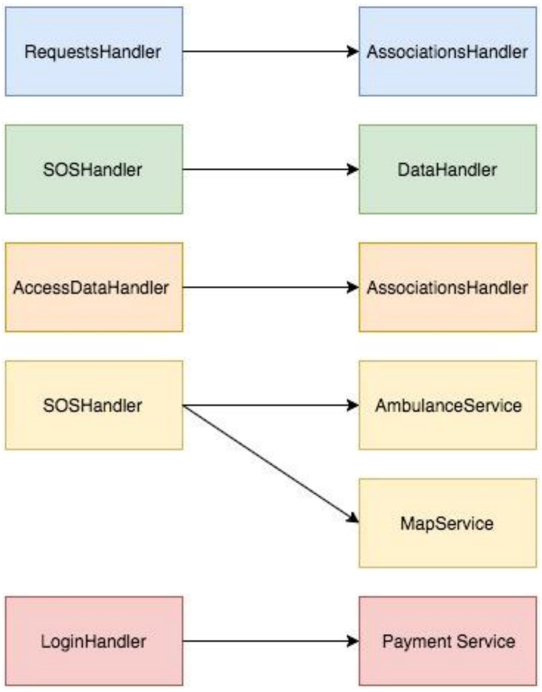
1) Dispatcher 1 and Dispatcher2
2) DataHandler, LogInHandler and clients
3) AssociationsHandler, RequestsHandler, SOSHandler and Database
4) QueryHandler, AccessDataHandler

The components on the same level can be implemented in parallel, the ones on the following ones can be started only after the component they depend on has reached a certain percentage of implementation, with respect to the constraints defined before.

Components integration

As mentioned before, the components of the system should be integrated as soon as possible.
As soon as they are implemented, the interfaces towards the other components should be built as well, in order to reduce the risks of having to modify completely some parts of the systems in order to make them adaptable with the others.

Every component must be integrated with the dispatchers, apart from that the main interface to be built are the following:



Testing

As soon as the implementation of the single components reaches a certain point, some tests should be made.
The same goes for the test on the integration between the components, which should start before the components are completely implemented.
As mentioned before, particular attention should be payed to the test on the dispatchers, given the importance of the role they cover in the system.

6. EFFORT SPENT

Luca Grella

DATE	TASK	HOURS
16 Nov	Sequence Diagrams	3h
18 Nov	Sequence Diagrams	3h
22 Nov	Component Integration	1,5h
23 Nov	Component View	2h
24 Nov	Component Diagram	4h
27 Nov	Deploying View	3,5h
3 Dec	High Level Components	1h
5 Dec	Overview	2h
6 Dec	Document Structure	1h
7 Dec	Diagrams Review	2,5h
8 Dec	Diagrams Review	2h
9 Dec	RASD Review	2h
10 Dec	DD Review	3h

Tot. 30,5h

Daniele Lunghi

DATE	TASK	HOURS
16 Nov	Purpose, Scope, Acronyms	3h
18 Nov	Overview, Component View	3h
22 Nov	Deployment, Runtime views	4h
23 Nov	Design Patterns	4h
25 Nov	Requirements Traceability	1h
26 Nov	Test plan	1h
29 Nov	Implementation, Integration	4h
6 Dec	Component Interfaces	4h
7 Dec	Class and Interfaces Review	2h
8 Dec	DD Structure Review	2h
9 Dec	RASD Review	2h
10 Dec	DD Review	3h

Tot. 33h

7. REFERENCES

1) Draw.io
2) GitHubDesktop 1.0.6
3) AdobePhotoshop CC 2017
4) StarUml 2.8.0